Artificial Intelligence 206 (2014) 79–111

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Algorithm runtime prediction: Methods & evaluation

Frank Hutter *, Lin Xu, Holger H. Hoos, Kevin Leyton-Brown

Department of Computer Science, University of British Columbia, 201-2366 Main Mall, BC V6T 1Z4, Canada

ARTICLE INFO

Article history:
Received 6 November 2012
Received in revised form 12 October 2013
Accepted 18 October 2013
Available online 24 October 2013

Keywords:
Supervised machine learning
Performance prediction
Empirical performance models
Response surface models
Highly parameterized algorithms
Propositional satisfiability
Mixed integer programming
Travelling salesperson problem

ABSTRACT

Perhaps surprisingly, it is possible to predict how long an algorithm will take to run on a previously unseen input, using machine learning techniques to build a model of the algorithm's runtime as a function of problem-specific instance features. Such models have important applications to algorithm analysis, portfolio-based algorithm selection, and the automatic configuration of parameterized algorithms. Over the past decade, a wide variety of techniques have been studied for building such models. Here, we describe extensions and improvements of existing models, new families of models, and—perhaps most importantly—a much more thorough treatment of algorithm parameters as model inputs. We also comprehensively describe new and existing features for predicting algorithm runtime for propositional satisfiability (SAT), travelling salesperson (TSP) and mixed integer programming (MIP) problems. We evaluate these innovations through the largest empirical analysis of its kind, comparing to a wide range of runtime modelling techniques from the literature. Our experiments consider 11 algorithms and 35 instance distributions; they also span a very wide range of SAT, MIP, and TSP instances, with the least structured having been generated uniformly at random and the most structured having emerged from real industrial applications. Overall, we demonstrate that our new models yield substantially better runtime predictions than previous approaches in terms of their generalization to new problem instances, to new algorithms from a parameterized space, and to both simultaneously.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

NP-complete problems are ubiquitous in AI. Luckily, while these problems may be hard to solve on worst-case inputs, it is often feasible to solve even large problem instances that arise in practice. Less luckily, state-of-the-art algorithms often exhibit extreme runtime variation across instances from realistic distributions, even when problem size is held constant, and conversely the same instance can take dramatically different amounts of time to solve depending on the algorithm used [31]. There is little theoretical understanding of what causes this variation. Over the past decade, a considerable body of work has shown how to use supervised machine learning methods to build regression models that provide approximate answers to this question based on given algorithm performance data; we survey this work in Section 2. In this article, we refer to such models as *empirical performance models* (*EPMs*).[1] These models are useful in a variety of practical contexts:

* Corresponding author.
E-mail addresses: hutter@cs.ubc.ca (F. Hutter), xulin730@cs.ubc.ca (L. Xu), hoos@cs.ubc.ca (H.H. Hoos), kevinlb@cs.ubc.ca (K. Leyton-Brown).

[1] In work aiming to gain insights into instance hardness beyond the worst case, we have used the term *empirical hardness model* [75,76,73]. Similar regression models can also be used to predict objectives other than runtime; examples include an algorithm's success probability [45,97], the solution quality an optimization algorithm achieves in a fixed time [96,20,56], approximation ratio of greedy local search [82], or the SAT competition scoring function [119]. We reflect this broadened scope by using the term EPMs, which we understand as an umbrella that includes EHMs.

0004-3702/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.artint.2013.10.003

- **Algorithm selection.** This classic problem of selecting the best from a given set of algorithms on a per-instance basis [95,104] has been successfully addressed by using EPMs to predict the performance of all candidate algorithms and selecting the one predicted to perform best [18,79,26,45,97,119,70].
- **Parameter tuning and algorithm configuration.** EPMs are useful for these problems in at least two ways. First, they can model the performance of a parameterized algorithm dependent on the settings of its parameters; in a sequential model-based optimization process, one alternates between learning an EPM and using it to identify promising settings to evaluate next [65,7,59,55,56]. Second, EPMs can model algorithm performance dependent on both problem instance features and algorithm parameter settings; such models can then be used to select parameter settings with good predicted performance on a per-instance basis [50].
- **Generating hard benchmarks.** An EPM for one or more algorithms can be used to set the parameters of existing benchmark generators in order to create instances that are hard for the algorithms in question [74,76].
- **Gaining insights into instance hardness and algorithm performance.** EPMs can be used to assess which instance features and algorithm parameter values most impact empirical performance. Some models support such assessments directly [96,82]. For other models, generic feature selection methods, such as forward selection, can be used to identify a small number of key model inputs (often fewer than five) that explain algorithm performance almost as well as the whole set of inputs [76,57].

While these applications motivate our work, in the following, we will not discuss them in detail; instead, we focus on the models themselves. The idea of modelling algorithm runtime is no longer new; however, we have made substantial recent progress in making runtime prediction methods more general, scalable and accurate. After a review of past work (Section 2) and of the runtime prediction methods used by this work (Section 3), we describe four new contributions.

1. We describe new, more sophisticated modelling techniques (based on random forests and approximate Gaussian processes) and methods for modelling runtime variation arising from the settings of a large number of (both categorical and continuous) algorithm parameters (Section 4).
2. We introduce new instance features for propositional satisfiability (SAT), travelling salesperson (TSP) and mixed integer programming (MIP) problems—in particular, novel probing features and timing features—yielding comprehensive sets of 138, 121, and 64 features for SAT, MIP, and TSP, respectively (Section 5).
3. To assess the impact of these advances and to determine the current state of the art, we performed what we believe is the most comprehensive evaluation of runtime prediction methods to date. Specifically, we evaluated all methods of which we are aware on performance data for 11 algorithms and 35 instance distributions spanning SAT, TSP and MIP and considering three different problems: predicting runtime on novel instances (Section 6), novel parameter configurations (Section 7), and both novel instances *and* configurations (Section 8).
4. Techniques from the statistical literature on survival analysis offer ways to better handle data from runs that were terminated prematurely. While these techniques were not used in most previous work—leading us to omit them from the comparison above—we show how to leverage them to achieve further improvements to our best-performing model, random forests (Section 9).[2]

2. An overview of related work

Because the problems have been considered by substantially different communities, we separately consider related work on predicting the runtime of parameterless and parameterized algorithms, and applications of these predictions to gain insights into instance hardness and algorithm parameters.

2.1. Related work on predicting runtime of parameterless algorithms

The use of statistical regression methods for runtime prediction has its roots in a range of different communities and dates back at least to the mid-1990s. In the parallel computing literature, Brewer used linear regression models to predict the runtime of different implementations of portable, high-level libraries for multiprocessors, aiming to automatically select the best implementation on a novel architecture [17,18]. In the AI planning literature, Fink [26] used linear regression to predict how the performance of three planning algorithms depends on problem size and used these predictions for deciding which algorithm to run for how long. In the same community, Howe and co-authors [45,97] used linear regression to predict how both a planner's runtime and its probability of success depend on various features of the planning problem; they also applied these predictions to decide, on a per-instance basis, which of a finite set of algorithms should be run in order to optimize a performance objective such as expected runtime. Specifically, they constructed a *portfolio* of planners that ordered algorithms by their expected success probability divided by their expected runtime. In the constraint programming

[2] We used early versions of the new modelling techniques described in Section 4, as well as the extensions to censored data described in Section 9 in recent conference and workshop publications on algorithm configuration [59,55,56,54]. This article is the first to comprehensively evaluate the quality of these models.

Download English Version:

https://daneshyari.com/en/article/376940

Download Persian Version:

https://daneshyari.com/article/376940

Daneshyari.com